IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: **Levine et al.**	§	
	§	Group Art Unit: **2183**
Serial No. **10/674,606**	§	
	§	Examiner: **Ryan Paul Fiegle**
Filed: **September 30, 2003**	§	
	§	
For: **Method and Apparatus to Autonomically Generate Information on Calls and Returns in a Program**	§	

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

35525
PATENT TRADEMARK OFFICE
CUSTOMER NUMBER

APPEAL BRIEF (37 C.F.R. 41.37)

This brief is in furtherance of the Notice of Appeal, filed in this case on September 18, 2006.

A fee of $500.00 is required for filing an Appeal Brief. Please charge this fee to IBM Corporation Deposit Account No. 09-0447. No additional fees are believed to be necessary. If, however, any additional fees are required, I authorize the Commissioner to charge these fees which may be required to IBM Corporation Deposit Account No. 09-0447. No extension of time is believed to be necessary. If, however, an extension of time is required, the extension is requested, and I authorize the Commissioner to charge any fees for this extension to IBM Corporation Deposit Account No. 09-0447.

REAL PARTY IN INTEREST

The real party in interest in this appeal is the following party: International Business Machines Corporation of Armonk, New York.

RELATED APPEALS AND INTERFERENCES

With respect to other appeals or interferences that will directly affect, or be directly affected by, or have a bearing on the Board's decision in the pending appeal, Appellants note that the following related applications are currently under appeal with the U.S. Patent Office.

Application Serial Number 10/675,831, filed September 30, 2003, Attorney docket Number AUS920030479US1;

Application Serial Number 10/675,778, filed September 30, 2003, Attorney docket Number AUS920030480US1;

Application Serial Number 10/674,606, filed September 30, 2003, Attorney docket Number AUS920030485US1.

STATUS OF CLAIMS

A. TOTAL NUMBER OF CLAIMS IN APPLICATION

Claims in the application are: 1-24

B. STATUS OF ALL THE CLAIMS IN APPLICATION

1. Claims canceled: 4, 11, and 18
2. Claims withdrawn from consideration but not canceled: None
3. Claims pending: 1-3, 5-10, 12-17, and 19-24
4. Claims allowed: None
5. Claims rejected: 1-3, 5-10, 12-17, and 19-24
6. Claims objected to: None

C. CLAIMS ON APPEAL

The claims on appeal are: 1-3, 5-10, 12-17, and 19-24

STATUS OF MENDMENTS

No amendments have been submitted since the Final Office Action mailed June 23, 2006.

SUMMARY OF CLAIMED SUBJECT MATTER

A. CLAIM 1 - INDEPENDENT

As a non-limiting example, the subject matter of Claim 1 relates to monitoring the execution of a program by storing a respective indicator in an indicator location associated with each call and return in the program, and then executing the program to generate data on calls and returns in the program. Specifically, Claim 1 is directed to a method in a data processing system for monitoring the execution of a program (**Figure 1**, page 12, line 1 through page 14, line 16). The method includes, in a system having an indicator location associated with each instruction (**Figure 4**, page 25, line 6 through page 26, line 19), storing a respective indicator in the indicator location associated with each call and return in the program (**Figure 28, 2802**, page 58, lines 12-13); and executing the program using a processor, wherein the respective indicators associated with the calls and returns cause the processor executing the instructions to generate data on calls and returns in the program (**Figure 28, 2804**, page 57, lines 8-27, page 58, lines 13-16).

B. CLAIM 8 - INDEPENDENT

Claim 8 is directed to a data processing system for monitoring the execution of a program (**Figure 1**, page 12, line 1 through page 14, line 16). This claim is the data processing system counterpart to claim 1.

C. CLAIM 15 - INDEPENDENT

Claim 15 is directed to a computer program product for monitoring the execution of a program (page 64, line 15 through page 65, line 3). This claim is the computer program product counterpart to claim 1.

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

A. GROUND OF REJECTION 1

Whether the Office Action failed to state a *prima facie* obviousness rejection under 35 U.S.C. §103(a) of claims 1, 8, 15 and 22-24 over *Smolders*, System for Tracing Hardware Counters Utilizing Programmed Performance Monitor to Generate Trace Interrupt After Each Branch Instruction or at the End of Each Code Basic Block, U.S. Patent 6,253,338 (June 26, 2001) (hereinafter *Smolders*) in view of *Buser*, Software Breakpoints with Tailoring for Multiple Processor Shared Memory or Multiple Thread Systems, U.S. Patent Application Publication 2004/0030870 (February 12, 2004) (hereinafter *Buser*).

B. GROUND OF REJECTION 2

Whether the Office action failed to state a *prima facie* obviousness rejection under 35 U.S.C. §103(a) of claims 2-7, 9-14, and 16-21 over *Smolders*, *Buser*, and *Subrahmanyam*, Transparent Instrumentation for Computer Program Behavior Analysis, U.S. Patent 5,987,250 (November 16, 1999) (hereinafter "*Subrahmanyam*").

ARGUMENT

A. GROUND OF REJECTION 1 (Claims 1, 8, 15, and 22-24)

Claim 1 is representative of the claims in this grouping of claims. Claim 1 is as follows:

> 1. A method in a data processing system for monitoring the execution of a program, the method comprising:
>
> in a system having an indicator location associated with each instruction, storing a respective indicator in the indicator location associated with each call and return in the program; and
>
> executing the program using a processor, wherein the respective indicators associated with the calls and returns cause the processor executing the instructions to generate data on calls and returns in the program.

Regarding claim 1, the rejection states:

> Smolders teaches a method in a data processing system for monitoring the execution of a program, the method comprising:
>
> associating instructions for calls and returns in the program with a set of indicators (column 4, lines 15-18; column 3, lines 29-33; column 5, lines 20-24; column 5, lines 39-43) (Smolders associates all branches with indicators, this includes calls and returns. The indicators are T, the process information and counters.); and
>
> executing the program using a processor, wherein the set of indicators associated with the instructions causes the processor executing the instructions to generate data on calls and returns in the program (column 4, lines 18-21; column 4, lines 60-65).
>
> Smolders does not teach his indicators being in a field associated with each instruction for holding a potential indicator, while Buser does (Buser: Abstract; Figure 1).
>
> Buser states that debugging problems arise in shared memory systems because not all processors are able to honor a breakpoint because they are not aware of it. Buser's method solves these deficiencies (Buser 0002, 0003).
>
> Therefore, it would have been obvious to one of the ordinary skill in the pertinent art that Buser's indicator field would be advantageous when using Smolder's debugging method in a shared memory system.

Final Office Action mailed June 23, 2006, page 3.

A.1. The Proposed Combination When Considered as a Whole Does Not Teach or Suggest All of the Features of the Claims

The determination of "nonobviousness" is made after establishing the scope and content of prior art, the differences between the prior art and the claims at issue, and the level of ordinary skill in the pertinent art. *Graham v. John Deere*, 383 U.S. 1 (1966). In addition, all limitations of the claimed invention must be considered when determining patentability. *In re Lowry*, 32 F.3d 1579, 1582, 21 U.S.P.Q.2d 1031, 1034 (Fed Cir. 1994). In the case at hand, the Office Action has not properly considered all features of Claim 1.

Specifically, the Office Action has failed to state a *prima facie* obviousness rejection because the proposed combination of *Smolders* and *Buser*, when considered as a whole, does not teach all of the features of Claim 1. Specifically, none of *Smolders*, *Buser*, or a combination of these references teaches or suggests the step of "storing a respective indicator in the indicator location associated with each call and return in the program," as provided in Claim 1.

The Office Action asserts otherwise, citing *Smolders* as teaching this claimed feature. Specifically, the Office Action cites *Smolders* as follows.

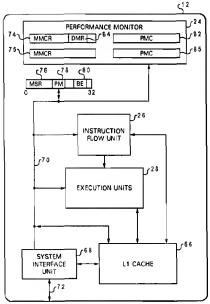

Fig. 2

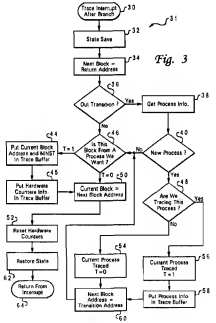

Fig. 3

As shown in FIG. 2, within the performance monitor 24 are monitor mode control registers (MMCR) 74 and 75, respectively, used for programming

> and one or more associated performance monitor hardware counters (PMC) 82 and 85 that used for counting operations. ... Lastly, the instruction flow unit 26 is programmed to generate a trace interrupt after each branch by setting a specified branch trace enable bit 80 in the machine state register 76. ... When a branch is found, the flow of execution proceeds as described below in FIG. 3, if the instruction was not a branch, the flow of execution simply returns to the next instruction in sequence without any additional action.

Smolders, column 3, line 29 through column 4, line 11.

> As described above, the instruction flow unit of the system processor unit 12 generates a trace interrupt after each branch instruction or at the end of each basic block of code as shown in step 30.

Smolders, column 4, lines 15-18.

> Step 56 shows that if the current process is to be traced then a variable T is set equal to one wherein the next step 58 puts the process information, (for example, the PID, TID and program name) into a trace buffer and continues to step 60.

Smolders, column 5, lines 20-24.

> If it is then the current basic block address (i.e. tracing information) and the value of the hardware counters, 74 and 75, respectively, are placed in the trace buffer, as shown in steps 44 and 45, and the counter level tracing tool 31 continues to step 50.

Smolders, column 5, lines 39-43.

However, the Office Action's characterization of *Smolders* is incorrect. *Smolders* teaches generating an interrupt after each branch instruction. The Office Action asserts that this method would include calls and returns. Based on this assertion, the Office Action concludes that *Smolders* discloses the claimed feature of, "*storing a respective indicator ... associated with each call and return in the program*". However, this interpretation ignores the difference between a branch and a call or return. In fact, *Smolders* does not teach the feature of Claim 1, as asserted by the Office Action.

In the following discussion, Appellants will refer to *call* instructions, but one of ordinary skill will understand that similar arguments apply also to *return* instructions. Whenever a *call* instruction is performed, at least two actions occur: (1) a return address is pushed onto the call stack and (2) control is transferred to the subroutine. In effect, a call instruction behaves as a store

instruction combined with a transfer of control instruction. A *branch* instruction, in contrast, is only a transfer of control. This fact is supported by the definitions enclosed in the appendix as evidence. For example, the technical dictionary published by International Business Machines Corporation (IBM) provides the following definitions for *branch* and *call*:

> **branch** 1. In the execution of a computer program, to select one from a number of alternative sets of instructions.
> 2. In a network, a path that connects two adjacent nodes and that has no intermediate nodes.
> 3. A set of instructions that are executed between two successive branch instructions.
> 4. Loosely, a conditional jump.
> 5. To select a branch as in (3).
> 6. Deprecated term for jump.
>
> **call** 1. The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point.
> 2. To transfer control to a procedure, program, routine, or subroutine.
> 3. In data communication, the actions necessary to make a connection between two stations.
> 4. To attempt to contact a user, regardless of whether the attempt is successful.
> 5. Synonymous with cue.
> 6. See procedure call, subroutine call.

Dictionary of Computing, published by International Business Machines Corporation, 1987.

These definitions show that a *branch* has a different meaning in the art than a *call*. Another reference provides further evidence of the different status of these instructions in the art. At this instruction level, machine language rather than a higher-level programming language, a *branch* is referred to as a *jump* instruction.

> The JMP (jump) instruction unconditionally transfers program control to a destination instruction. The transfer is one-way; that is, a return address is not saved. A destination operand specifies the address (the instruction pointer) of the destination instruction. The address can be a **relative address** or an **absolute address**.
>
> The CALL instruction transfers program control from the current (or calling procedure) to another procedure (the called procedure). To allow a subsequent return to the calling procedure, the CALL instruction saves the

> current contents of the EIP register on the stack before jumping to the called procedure. The EIP register (prior to transferring program control) contains the address of the instruction following the CALL instruction. When this address is pushed on the stack, it is referred to as the **return instruction pointer** or **return address**.

Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference, Intel order number 243191, pages 6-35, 6-36 (emphasis added).

As these definitions demonstrate, one of ordinary skill in the art understands that a *call* is different than a *branch* and that the recitation of storing an indicator to every *call*, as claimed, is not encompassed by storing an indicator to every *branch*, as disclosed in *Smolders*. Thus, *Smolders* does not teach the claimed feature of, "storing a respective indicator in the indicator location associated with each call and return in the program," as in Claim 1. Because the claimed feature is different than that disclosed in the reference, and because *Smolders* is devoid of disclosure in this regard, *Smolders* also does not suggest this claimed feature.

Additionally, *Buser* does not teach or suggest this claimed feature. The Office Action does not assert that *Buser* teaches or suggests this claimed feature. Additionally, *Buser*, discussed below, actually does not teach or suggest this claimed feature. Therefore, neither the references relied on nor their combination teach or suggest the step of, "storing a respective indicator in the indicator location associated with each call and return in the program", as in Claim 1.

Additionally, neither the references relied on nor their combination teach or suggest the step of "executing the program using a processor, wherein the respective indicators associated with the calls and returns cause the processor executing the instructions to generate data on calls and returns in the program". The specification on page 58 discusses some of the "data on calls and returns" that can be collected using the invention recited in claim 1, such as "identification of callers of routines", which can then be "used to generate data structures, such as trees to track and present information regarding the execution of the program" (application, page 58, third full paragraph). The information generated by the method recited in Claim 1 is specifically "data on calls and returns;" the information is not data generation that is triggered by a call or return. In contrast, *Smolders* is only capable of counting events that occur between branch points. Exemplary data collected by *Smolders* is disclosed to be "counting the number of cycles during a

selected executing process or the number of load/store misses occurring within an L2 cache" (*Smolders*, column 3, lines 41-44). Thus, *Smolders* does not disclose a capability to provide "*data on calls and returns*", as recited in Claim 1. *Buser* also does not disclose generating the claimed "*data on calls and returns*", nor does the rejection assert that this reference does so. Therefore, none of *Smolders*, *Buser*, or a combination of the two references teach or suggest the claimed feature of, "wherein the respective indicators associated with the calls and returns cause the processor executing the instructions to generate data on calls and returns in the program," as in Claim 1.

Appellants have proved that none of *Smolders*, *Buser*, or their combination teach or suggest either of the steps of the invention recited in Claim 1. Therefore, the Office Action has failed to state a *prima facie* obviousness rejection against Claim 1 or against any other claim in this grouping of claims.

A.2. The Office Action Failed To State a Proper Teaching, Suggestion, or Motivation To Combine the References To Achieve the Invention of the Claims

In addition, the Office Action has failed to state a *prima facie* obviousness rejection against Claim 1 because the Office Action has not stated a proper teaching, suggestion, or motivation to combine the references. Instead, the Office Action has only stated a proposed advantage to combining the references. However, an advantage is not necessarily a teaching, suggestion, or motivation.

To constitute a proper teaching, suggestion, or motivation, the Office Action must establish that one of ordinary skill would both recognize the advantage and have a reason to implement the advantage. For example, a first reference may disclose the use of lasers to achieve nuclear fusion. A second reference may disclose that ultra-high power lasers deliver more energy. One of ordinary skill may recognize that an ultra-high power laser would be more useful to achieve nuclear fusion. However, one of ordinary skill would be motivated to avoid combining the references because of the extreme expense of ultra-high power lasers. In this example, one of ordinary skill is motivated to avoid implementing the combination, even if he or she recognized the advantage. Therefore, in this example, no teaching, suggestion, or motivation exists to combine the references.

Similarly, in the case at hand, the Office Action has not provided a sufficient reason why one of ordinary skill would have a reason to implement the proposed advantage. The Office Action states that "debugging problems arise in shared memory systems because not all processors are able to honor a breakpoint because they are not aware of it. *Buser's* method solves these deficiencies (Buser 0002, 0003). *Buser's* indicator field would be advantageous when using *Smolder's* debugging method in a shared memory system." However, the proposed motivation does not actually exist because *Smolder* is not concerned with honoring a common breakpoint among multiple processors. Because no need exists for a common breakpoint, the Office Action has established no reason why one of ordinary skill would recognize any advantage to combining the references. For these reasons, the Office Action's statement fails to provide a proper teaching, suggestion, or motivation to combine the references. Accordingly, the Office Action has failed to state a *prima facie* obviousness rejection against claim 1 or against any other claim in this grouping of claims.

A.3. No Motivation Exists to Combine *Smolders* and *Buser* Because They Address Different Problems

One of ordinary skill would not combine the references to achieve the invention of Claim 1 because the references are directed towards solving different problems. It is necessary to consider the reality of the circumstances--in other words, common sense--in deciding in which fields a person of ordinary skill would reasonably be expected to look for a solution to the problem facing the inventor. *In re Oetiker*, 977 F.2d 1443 (Fed. Cir. 1992); *In re Wood*, 599 F.2d 1032, 1036, 202 U.S.P.Q. 171, 174 (CCPA 1979). In the case at hand, the cited references address distinct problems. Thus, no common sense reason exists to establish that one of ordinary skill would reasonably be expected to look for a solution to the problem facing the inventor. Accordingly, no teaching, suggestion, or motivation exists to combine the references and the Office Action has failed to state a *prima facie* obviousness rejection of Claim 1.

For example, *Smolders* is directed to solving the problem of reducing overhead costs of using API calls while maintaining higher granularity. For example, *Smolders* provides that:

> Application Programming Interfaces (API) have also been built to collect counter information for portions of workloads. In this mode, one must add calls to API code just before and immediately after the execution of the

> portion of the workload to analyze. The former is to setup and start the counting and the latter is to stop the counting and retrieve the results. Those API calls can either be added directly to the source code if it is available or by way of instrumentation, i.e. dynamic insertion of code to divert normal execution path. This technique provides lower granularity results than the global counting of the previous approach, but at a big cost due to the overhead of the outside code necessary to control the counting. The overhead is what directly limits the obtainable granularity of the results.
>
> Consequently, it would be desirable to provide an improved method and system that determines which part of a workload is responsible for counter increments of desired events without any overhead. In particular, it would be desirable to provide an improved method and system which utilizes a performance monitor facility and generates an exception after each branch instruction for gaining control at a basic block level for counting various events happening on a microprocessor.

Smolders, col. 1, ll. 33-55.

On the other hand, *Buser* is directed to the problem of reducing debugging problems when common shared memory contains executable code. For example, *Buser* provides as follows:

> [0002] The advent of the system-on-a-chip (SOC) architectures for embedded systems has created many challenges for the software development systems used to develop and debug software applications that execute on these architectures. These systems may be comprised of multiple interconnected processors that share the use of on-chip and off-chip memory. A processor may include some combination of instruction cache (ICache) and data cache (DCache) to improve processing performance and can be instantiated from a design library as a single megacell. Furthermore, multiple megacells, with memory being shared among them, may be incorporated in a single embedded system. The processors may physically share the same memory without accessing data or executing code located in the same memory locations or they may use some portion of the shared memory as common shared memory.
>
> [0003] Common shared memory contains executable code or data that will be accessed or executed by more than one processor, possibly simultaneously. While such memory sharing is advantageous in the final product, it creates potential debugging problems during application development. If a software breakpoint (SWBP) is set in common shared memory, all processors that access that shared memory location must honor the breakpoint. Also, if one processor halts at the breakpoint, any other

processor that could potentially access that shared memory location should also be halted to ensure that no breakpoints are missed and invalid code is not executed.

Buser, paragraphs 0002 through 0003.

Based on the plain disclosures of the references themselves, the references address completely distinct problems that are unrelated to each other. The problem of reducing overhead costs of using API calls while maintaining higher granularity is completely distinct from the problem of reducing debugging problems when common shared memory contains executable code.

Because the references address completely distinct problems, one of ordinary skill would have no reason to combine or otherwise modify the references to achieve the invention of Claim 1. Thus, no proper teaching, suggestion, or motivation exists to combine the references in the manner suggested by the Office Action. Accordingly, the Office Action has failed to state a *prima facie* obviousness rejection against Claim 1 or any other claim in this grouping of claims.

B. GROUND OF REJECTION 2 (Claims 2-7, 9-14, and 16-21)

The Office Action rejects claims 2-7, 9-14, and 16-21 as obvious in view of *Smolders*, *Buser*, and *Subrahmanyam*. The Office Action states that:

> Smolders does not teach the following limitations which Subrahmanyam does:
>
> responsive to identifying an instruction in an instruction cache for execution during execution of the program, determining whether an indicator from the set of indicators is associated with the instruction (Subrahmanyam: column 4, lines 9-17; column 4, lines 35-28) (A probe location is marked by a flag indicator which is associated with a location in the instruction code, the location being an instruction, which will inherently come from an instruction cache since Subrahmanyam contains an instruction cache (Subrahmanyam: column 3, lines 60-64); and
>
> generating an interrupt if the indicator is associated with the instruction, wherein the interrupt causes execution of a program to generate data on the calls and returns in the program (Subrahmanyam: column 4, lines 38-43; column 1, lines 12-17; column 4, lines 2-5) (Subrahmanyam states that various analysis tools are well known in the art which his method can be used for. One of these tools is analyzing calls and returns.).

> Smolders explicitly inserts monitor code into the instruction stream (Smolders: column 4, lines 21-26). Subrahmanyam states that this is undesirable for several reasons (Subrahmanyam: column 1, lines 50-65). Subrahmanyam's system collects data throughout regular execution, accumulating the data. Then when a probe location is reached, analysis is done and the data is dumped into an external file. When applying Smolders is applied to Subrahmanyam, data would be collected on calls and returns as normal, but it would be accumulated until a probe location is reached, rather than interrupting to a analysis tool within the same process after each branch. Doing this allow Smolders to observe call and return behavior without, "affecting program behavior," (Subrahmanyam: column 2, lines 1-3) and would avoid the pitfalls observed by Subrahmanyam (Subrahmanyam: column 1, lines 50-65).
>
> Therefore, it would have been obvious to one of ordinary skill in the pertinent art at the time of the applicant's invention that applying Subrahmanyam to Smolders would allow for studying program behavior without affecting the program behavior.

Final Office Action of June 23, 2006, pp. 4-6.

B.1. Claims 2, 6, 9, 13, 16, 20

Claim 2 is a representative claim in this grouping of claims. Claim 2 is as follows:

> 2. The method of claim 1 further comprising:
> responsive to identifying an instruction in an instruction cache for execution during execution of the program, determining whether an indicator is stored in the respective indicator location associated with the instruction; and
> generating an interrupt if the indicator is stored in the respective indicator location associated with the instruction, wherein the interrupt causes execution of a program to generate data on the calls and returns in the program.

B.1.i. The Office Action Failed To State a Proper Teaching, Suggestion, or Motivation To Combine the References

With regard to a teaching, suggestion, or motivation to combine the references, the Office Action states that:

> Therefore, it would have been obvious to one of ordinary skill in the pertinent art at the time of the applicant's invention that applying Subrahmanyam to Smolders would allow for studying program behavior without affecting the program behavior.

Final Office action of June 23, 2006, p. 6.

On its face, the rejection fails to provide a teaching, suggestion, or motivation to combine *Smolders*, *Subrahmanyam*, ***and*** *Buser*. Instead, the Office Action on refers to the combination of *Smolders* and *Subrahmanyam*. Under the standards of *Graham v. John Deere*, the examiner is required to provide a proper teaching, suggestion, or motivation to combine all of the reference together as a whole. By ignoring *Buser* the Office Action has failed to comply with this requirement. Therefore, the Office Action has failed to state a *prima facie* obviousness rejection against Claim 2 and the other claims in this grouping of claims.

B.1.ii. The Combination of the References, When Considered as a Whole, Does Not Teach or Suggest All of the Features of Claim 2

As shown above, with respect to Claim 1, the proposed combination of *Smolders* and *Buser*, when considered as a whole, does not teach or suggest all of the features of Claim 1. Claim 2 depends from Claim 1. Therefore, at least by virtue of the dependency of Claim 2 from Claim 1, the proposed combination of *Smolders* and *Buser*, when considered as a whole, does not teach or suggest all of the features of Claim 2 for the reasons given above. Accordingly, the Office Action has failed to state a *prima facie* obviousness rejection against Claim 2 and the other claims in this grouping of claims.

B.1.iii. No Motivation Exists to Combine *Smolders*, *Buser*, and *Subrahmanyam* Because They Address Different Problems

One of ordinary skill would not combine the references to achieve the invention of Claim 2 because the references are directed towards solving different problems. It is necessary to consider the reality of the circumstances--in other words, common sense--in deciding in which fields a person of ordinary skill would reasonably be expected to look for a solution to the problem facing the inventor. *In re Oetiker*, 977 F.2d 1443 (Fed. Cir. 1992); *In re Wood*, 599 F.2d 1032, 1036, 202 U.S.P.Q. 171, 174 (CCPA 1979). In the case at hand, the cited references address distinct problems. Thus, no common sense reason exists to establish that one of ordinary skill would

reasonably be expected to look for a solution to the problem facing the inventor. Accordingly, no teaching, suggestion, or motivation exists to combine the references and the Office Action has failed to state a *prima facie* obviousness rejection of Claim 2.

For example, *Smolders* is directed to solving the problem of reducing overhead costs of using API calls while maintaining higher granularity. In contrast, *Buser* is directed to the problem of reducing debugging problems when common shared memory contains executable code. In still further contrast, *Subrahmanyam*, is directed to the problem of creating a non-intrusive means of instrumenting an application program without modifying the application program code. For example, *Subrahmanyam* provides that:

> The Srivastava system creates a single, modified executable file such that the application program and the user's analysis routines run together in the same address space. See FIG. 1. Information is directly passed from the application program to the analysis routines, through simple procedure calls instead of inter-process communication. The system takes care that analysis routines do not interfere with the program's execution, but because it operates on the same CPU in the same address space, it necessarily affects the system performance. Moreover, that type of system will pollute the machine's resources, such as the instruction cache. Consequently, any behavior of the instruction cache will not be analyzable precisely. Additionally, since the instrumentation segments must be executed along with the application, it bloats the execution time of the application tremendously. Cases of 10 to 20-fold degradation in performance are not uncommon. Finally, the application must be protected from the instrumentation probes, since the probes will certainly change the register state of the machine. Hence, each probe must be encapsulated into a protective shield code. This shield code is quite "expensive."
>
> In view of the foregoing background, a principal object of the present invention is to provide a non-intrusive means for instrumenting an application program. A goal of the present invention is to allow for studying program behavior without affecting the program behavior. That is, an object of the present invention is to allow instrumenting an application program without making any modification to the application program code, or modifying the code in a way that is transparent during execution.
>
> Another object of the present invention is to provide for flexibility and customizing program instruments, by allowing a user to write custom "probes" as needed. A further object of the present invention is to provide for instrumenting the execution of an application program without affecting the machine's resources so that the program analysis is completely accurate.

Subrahmanyam, col. 1, l. through col. 2, l. 14.

Based on the plain disclosures of the references themselves, the references address completely distinct problems that are unrelated to each other. The problem of reducing overhead costs of using API calls while maintaining higher granularity is completely distinct from the problem of reducing debugging problems when common shared memory contains executable code. In still further contrast, these problems are wholly unrelated to the problem of creating a non-intrusive means of instrumenting an application program without modifying the application program code.

Because the references address completely distinct problems, one of ordinary skill would have no reason to combine or otherwise modify the references to achieve the invention of Claim 2. Thus, no proper teaching, suggestion, or motivation exists to combine the references in the manner suggested by the Office Action. Accordingly, the Office Action has failed to state a *prima facie* obviousness rejection against claim 2 or any other claim in this grouping of claims.

B.2. Claims 3, 5, 7, 10, 12, 14, 17, 19, 21

Claim 3 is a representative claim in this grouping of claims. Claim 3 is as follows:

> 3. The method of claim 1, wherein execution of an instruction having an indicator stored in the indicator location associated with the instruction causes passing of control to at least one of a process that records calls and returns and a process that identifies a calling routine.

B.2.i. The Office Action Failed To State a Proper Teaching, Suggestion, or Motivation To Combine the References

With regard to a teaching, suggestion, or motivation to combine the references, the Office Action states that:

> Therefore, it would have been obvious to one of ordinary skill in the pertinent art at the time of the applicant's invention that applying Subrahmanyam to Smolders would allow for studying program behavior without affecting the program behavior.

Final Office Action of June 23, 2006, p. 6.

On its face, the rejection fails to provide a teaching, suggestion, or motivation to combine *Smolders*, *Subrahmanyam*, ***and*** *Buser*. Instead, the Office Action on refers to the combination of *Smolders* and *Subrahmanyam*. Under the standards of *Graham v. John Deere*, the Office Action is

required to provide a proper teaching, suggestion, or motivation to combine all of the reference together as a whole. By ignoring *Buser* the Office Action has failed to comply with this requirement. Therefore, the Office Action has failed to state a *prima facie* obviousness rejection against Claim 3 and the other claims in this grouping of claims.

B.2.ii. The Combination of the References, When Considered as a Whole, Does Not Teach or Suggest All of the Features of Claim 3

B.2.ii.a. The Proposed Combination Does Not Teach "Generate Data on Calls and Returns in the Calling Routine" as Recited in Claim 3

Under the standards of *In re Lowry*, the Office action failed to state a *prima facie* obviousness rejection against Claim 3 because the proposed combination, when considered as a whole, does not teach or suggest the claimed features of, "storing respective indicators in the indicator locations associated with instructions in the *calling routine*," and "wherein the respective indicators associated with the instructions causes the processor executing the instructions in the calling routine to generate data on calls and returns in the *calling routine*," as in Claim 3.

The Office Action asserts otherwise, stating that:

> The method of claim 1, wherein execution of an instruction associated with an indicator in the set of indicators causes passing of control to a process that records calls and returns (Subrahmanyam: column 4, lines 38-43; column 1, lines 12-17; column 4, lines 2-5).

Final Office Action of June 23, 2006, p. 6.

However, the Office Action's characterization of *Subrahmanyam*, vis-à-vis Claim 3, is incorrect. For example, the first cited portion of *Subrahmanyam* is as follows:

> Execution of the program proceeds as follows. After execution of each instruction, 62, the translation device checks whether the present location in the code is a probe location 64. *If a probe location is detected, execution halts, step 66, and the system creates an instance of the associated software probe, step 70, and stores the created probe instance in a queue, step 72. The probe instance queue may be a separate file 74. Next, execution of the application program continues, 76.* If execution is not yet completed, test 80, it proceeds via loop 82 to execution of the next instruction. Execution continues in this fashion so that, each time a probe location is detected, the process calls for creating an instance of the software probe associated with

> the detected probe location, without changing a current state of the execution of the application program. In other words, creation of the sequence of software probes is separate from the execution of the application program, for example being carried out by a separate processor having its own local memory, register stack, etc. After all of the instructions in the marked application program have been executed, execution terminates at step 82.

Subrahmanyam, col. 4, ll. 35-55 (emphasis showing portions cited by the Office Action).

The cited portion of *Subrahmanyam* teaches that probe locations are present within the code. If a probe location is detected, execution halts and the system creates an instance of the probe. Thus, a sequence of software probes is created apart from the execution of the application program.

However, these teachings are wholly irrelevant to the claimed feature of, "storing respective indicators in the indicator locations associated with instructions in the *calling routine*," as in Claim 3. *Subrahmanyam* is devoid of disclosure regarding indicator locations associated with instructions in the *calling routine*. The probes in *Subrahmanyam* are not equivalent to "indicators" as claimed in Claim 3. Thus, the cited portion of *Subrahmanyam* does not teach this claimed feature. Because *Subrahmanyam* is devoid of disclosure in this regard, nothing in *Subrahmanyam* suggests all of the features of Claim 3.

Nevertheless, the Office Action also cites the following portion of *Subrahmanyam* as teaching this claimed feature:

> *Programming tools are important for analyzing the performance of computer programs. A variety of program analysis tools are known. Examples include call graph based profiling tools, instruction profiling tools, system call summary tools, dynamic instruction counters, cache memory modeling, input/output summary tools, load and store counters, etc.* These tools help computer architects, for example, evaluate how well programs perform on new hardware platforms. Software authors use such tools to analyze their programs and identify critical pieces of code. Compiler writers often use such tools to find out how well their instruction scheduling or branch prediction algorithms perform.

Subrahmanyam, col. 1, ll. 12-23 (emphasis showing portions cited by the Office Action).

This portion of *Subrahmanyam* teaches that programming tools for analyzing programs are known. *Subrahmanyam* cites a number of known analyzing tools, such as call graph based profiling tools. However, this portion of *Subrahmanyam* is devoid of disclosure regarding

indicator locations associated with instructions in the *calling routine*, as in Claim 3. The fact that a call graph based profiling tool exists provides no indication that indicator locations be associated with instructions in the *calling routine*. Therefore, this cited portion of *Subrahmanyam* does not teach all of the features of Claim 3. Because *Subrahmanyam* is devoid of disclosure in this regard, nothing in *Subrahmanyam* suggests all of the features of Claim 3.

Finally, the Office Action also cites the following portion of *Subrahmanyam* as teaching the feature of, "storing respective indicators in the indicator locations associated with instructions in the *calling routine*," as in Claim 3:

> FIG. 2 is a simplified block diagram of a new process for instrumenting an application program according to the present invention. Referring to FIG. 2, a user creates a desired instrument by defining a set of software probes. *Each software probe is a segment of code which may be written for example, in the C language, that in general gathers information about the execution of a program.* Various instrumentation routines or probes are known in the prior art. By placing probes at selected locations in a program, the user can instrument the program to accomplish such tasks as block counting, cache simulation, counting load and store instructions, etc. The user also specifies locations in the application program object file which we will call probe locations, step 52 in FIG. 2.

Subrahmanyam, col. 1, ll. 12-23 (emphasis showing portions cited by the Office Action).

This portion of *Subrahmanyam* teaches *Subrahmanyam's* process of instrumenting an application program. *Subrahmanyam* specified that a number of user-defined probes are created. Each software probe is a segment of code written in a programming language that gathers information about the execution of a program. However, this portion of *Subrahmanyam* is devoid of disclosure regarding indicator locations associated with instructions in the *calling routine*, as in Claim 3. The fact that probes can be inserted into program code is not equivalent to the claimed feature, and the Office Action has failed to provide any basis that the claimed feature and the probes in *Subrahmanyam* are in any way equivalent. Therefore, this cited portion of *Subrahmanyam* does not teach all of the features of Claim 3. Because *Subrahmanyam* is devoid of disclosure in this regard, nothing in *Subrahmanyam* suggests all of the features of Claim 3.

The Office Action does not cite *Smolders* and *Buser* with respect to the claimed feature of, "storing respective indicators in the indicator locations associated with instructions in the *calling*

routine" in the rejection of Claim 3. Additionally, these two references fail to teach or suggest this claimed feature.

None of *Smolders*, *Buser*, or *Subrahmanyam* teach or suggest all of the features of Claim 3. Therefore, the proposed combination of these references, when considered as a whole, does not teach or suggest all of the features of Claim 3. Accordingly, the Office Action has failed to state a *prima facie* obviousness rejection against Claim 3 and the remaining claims in this grouping of claims.

B.2.ii.b. Dependency of Claim 3 from Claim 1

As shown above, with respect to claim 1, the proposed combination of *Smolders* and *Buser*, when considered as a whole, does not teach or suggest all of the features of Claim 1. Claim 3 depends from Claim 1. Therefore, at least by virtue of the dependency of Claim 3 from Claim 1, the proposed combination of *Smolders* and *Buser*, when considered as a whole, does not teach or suggest all of the features of Claim 3 for the reasons given above. Accordingly, the Office Action has failed to state a *prima facie* obviousness rejection against Claim 3 and the other claims in this grouping of claims.

B.2.iii. No Motivation Exists to Combine *Smolders*, *Buser*, and *Subrahmanyam* Because They Address Different Problems

One of ordinary skill would not combine the references to achieve the invention of Claim 3 because the references are directed towards solving different problems. In the case at hand, the cited references address distinct problems. Thus, no common sense reason exists to establish that one of ordinary skill would reasonably be expected to look for a solution to the problem facing the inventor. Accordingly, no teaching, suggestion, or motivation exists to combine the references and the Office Action has failed to state a *prima facie* obviousness rejection of Claim 3.

For example, *Smolders* is directed to solving the problem of reducing overhead costs of using API calls while maintaining higher granularity. In contrast, *Buser* is directed to the problem of reducing debugging problems when common shared memory contains executable code. In still further contrast, *Subrahmanyam*, is directed to the problem of creating a non-intrusive means of instrumenting an application program without modifying the application program code.

Based on the plain disclosures of the references themselves, the references address completely distinct problems that are unrelated to each other. The problem of reducing overhead costs of using API calls while maintaining higher granularity is completely distinct from the problem of reducing debugging problems when common shared memory contains executable code. In still further contrast, these problems are wholly unrelated to the problem of creating a non-intrusive means of instrumenting an application program without modifying the application program code.

Because the references address completely distinct problems, one of ordinary skill would have no reason to combine or otherwise modify the references to achieve the invention of Claim 3. Thus, no proper teaching, suggestion, or motivation exists to combine the references in the manner suggested by the Office Action. Accordingly, the Office Action has failed to state a *prima facie* obviousness rejection against Claim 3 or any other claim in this grouping of claims.

C. CONCLUSION

As shown above, the Office Action has failed to state a *prima facie* obviousness rejection against any of the claims. Therefore, Appellants request that the Board of Patent Appeals and Interferences reverse the rejections. Additionally, Appellants request that the Board direct the Office Action to allow the claims.

/Theodore D. Fay III/
Theodore D. Fay III
Reg. No. 48,504
YEE & ASSOCIATES, P.C.
PO Box 802333
Dallas, TX 75380
(972) 385-8777

CLAIMS APPENDIX

The text of the claims involved in the appeal is as follows:

1. A method in a data processing system for monitoring the execution of a program, the method comprising:

in a system having an indicator location associated with each instruction, storing a respective indicator in the indicator location associated with each call and return in the program; and

executing the program using a processor, wherein the respective indicators associated with the calls and returns cause the processor executing the instructions to generate data on calls and returns in the program.

2. The method of claim 1 further comprising:

responsive to identifying an instruction in an instruction cache for execution during execution of the program, determining whether an indicator is stored in the respective indicator location associated with the instruction; and

generating an interrupt if the indicator is stored in the respective indicator location associated with the instruction, wherein the interrupt causes execution of a program to generate data on the calls and returns in the program.

3. The method of claim 1, wherein execution of an instruction having an indicator stored in the indicator location associated with the instruction causes passing of control to at least one of a process that records calls and returns and a process that identifies a calling routine.

5. The method of claim 3 further comprising:

storing respective indicators in the indicator locations associated with instructions in the calling routine; and

executing the program using a processor, wherein the respective indicators associated with the instructions causes the processor executing the instructions in the calling routine to generate data on calls and returns in the calling routine.

6. The method of claim 1, wherein the indicator locations are located in a shadow memory.

7. The method of claim 1 further comprising:

identifying a called routine.

8. A data processing system for monitoring the execution of a program, the data processing system comprising:

a processor connected to a memory, said processor and said memory being configured with an indicator location associated with each instruction;

storing instructions for storing respective indicators in the indicator locations associated with calls and returns in the program; and

executing instructions for executing the program using the processor, wherein the respective indicators associated with the instructions causes the processor executing the instructions to generate data on calls and returns in the program.

9. The data processing system of claim 8 further comprising:

determining instructions, responsive to identifying an instruction in an instruction cache for execution during execution of the program, for determining whether an indicator is stored in the indicator location associated with the instruction; and

generating instructions for generating an interrupt if the indicator is stored in the indicator location associated with the instruction, wherein the interrupt causes execution of a program to generate data on the calls and returns in the program.

10. The data processing system of claim 8, wherein execution of an instruction having an indicator stored in the indicator location associated with the instruction causes passing of control to one of a process that records calls and returns and a process that identifies a calling routine.

12. The data processing system of claim 8,

wherein execution of an instruction associated with an indicator in the set of indicators causes passing of control to a process that identifies a calling routine;

wherein the associating instructions are first associating instructions and further comprising:

second associating instructions for storing an indicator in the indicator location associated with instructions in the calling routine; and

executing instructions for executing the program using a processor, wherein the indicators stored in the indicator locations associated with the instructions causes the processor executing the instructions in the calling routine to generate data on calls and returns in the calling routine.

13. The data processing system of claim 8, wherein the indicator locations are located in a shadow memory.

14. The data processing system of claim 8 further comprising:

identifying instructions for identifying a called routine.

15. A computer program product in a recordable-type computer readable medium for monitoring the execution of a program, the computer program product comprising:

first instructions for storing respective indicators in the indicator locations associated with calls and returns in the program, wherein the system has an indicator location associated with each instruction; and

second instructions for executing the program using a processor, wherein the respective indicators stored in indicator locations associated with the instructions causes the processor executing the instructions to generate data on calls and returns in the program.

16. The computer program product of claim 15 further comprising:

third instructions, responsive to identifying an instruction in an instruction cache for execution during execution of the program, for determining whether an indicator is stored in the indicator location associated with the instruction; and

fourth instructions for generating an interrupt if the indicator is stored in the indicator location associated with the instruction, wherein the interrupt causes execution of a program to generate data on the calls and returns in the program.

17. The computer program product of claim 15, wherein execution of an instruction having an indicator stored in the indicator location associated with the instruction causes passing of control to one of a process that records calls and returns and a process that identifies a calling routine.

19. The computer program product of claim 17 further comprising:

fifth instructions for storing a respective indicator in the indicator locations associated with instructions in the calling routine; and

sixth instructions for executing the program using a processor, wherein the indicators stored in indicator locations associated with the instructions causes the processor executing the instructions in the calling routine to generate data on calls and returns in the calling routine.

20. The computer program product of claim 15, wherein the indicator locations are located in a shadow memory.

21. The computer program product of claim 15 further comprising:

seventh instructions for identifying a called routine.

22. The method of claim 1, wherein the respective indicators each include an element chosen from the group consisting of a flag, a tag field, a threshold, and a count field.

23. The data processing system of claim 8, wherein the respective indicators each include an element chosen from the group consisting of a flag, a tag field, a threshold, and a count field.

24. The computer program product of claim 15, wherein the respective indicators each include an element chosen from the group consisting of a flag, a tag field, a threshold, and a count field.

EVIDENCE APPENDIX

The following dictionary excerpts are presented as evidence.

Dictionary of Computing, International Business Machines Corporation, 1987, includes pages for "branch", "call", and "return".

Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference, 1997, pages 3.38 to 3.48, 3.241 to 3.251, 3.407 to 3.412.

RELATED PROCEEDINGS APPENDIX

The following related applications are currently under appeal. No decisions have been made in the related applications.

Application Serial Number 10/675,831, filed September 30, 2003, Attorney docket Number AUS920030479US1;

Application Serial Number 10/675,778, filed September 30, 2003, Attorney docket Number AUS920030480US1;

Application Serial Number 10/674,606, filed September 30, 2003, Attorney docket Number AUS920030485US1.